Proceedings of the 16th International Conference on Numerical Methods in Fluid Dynamics, To appear in "Lecture Notes in Physics",
Springer−Verlag, Heidelberg Germany.

Aspects (and Aspect Ratios) of Cartesian Mesh Methods

Marsha J. Berger
Courant Institute
New York, NY 10012

Michael J. Aftosmis
NASA Ames Research Center
Moffett Field, CA 94035

6−10 July, 1998
Arcachon, France

Aspects (and Aspect Ratios) of Cartesian Mesh Methods

Marsha Berger[+] and Michael Aftosmis[*]

[+]Courant Institute, New York, NY 10012
email: *berger@cims.nyu.edu*

[*]NASA Ames Research Center, Moffett Field, CA 94025
email: *aftosmis@nas.nasa.gov*

Key Words: GRID GENERATION, CARTESIAN MESHES, COMPLEX GEOMETRY, EMBEDDED BOUNDARIES

1 Introduction

Over the last decade, researchers have taken another look at the use of Cartesian meshes for grid generation around complicated configurations [Melton (96)], [Karman (95)], [Charlton and Powell (97)], [Landsberg and Boris (93)], [Pember et al. (91)]. This research has demonstrated the advantages of cut-cell Cartesian approaches in robustness, efficiency and automatibility. Although the arbitrarily small cut-cells which appear at intersections with the geometry may introduce accuracy and stability concerns, Cartesian methods present a viable path for mesh generation around complex geometry.

This paper presents a concise description of a revised grid generation algorithm for Cartesian meshes with embedded geometry. While conceptually straightforward, certain decisions make the process extremely efficient both in terms of memory requirements and execution speed. Adaptively refined meshes with millions of cells may be generated for very complex geometries within minutes on a desktop workstation. The paper also presents an approach for resolving geometry which splits Cartesian cells into distinct regions with unique flow states. It further examines the possibility of reducing the number of Cartesian cells in a domain through the use of anisotropic sub-division of Cartesian cells using a variety of configurations. A detailed example of the run-time performance of the grid generator is included.

1.1 Background

A comprehensive presentation of Cartesian mesh methods must consider a variety of topics. This paper focuses mainly on the generation of Cartesian meshes with embedded boundaries. This includes the computational geometry algorithms and data structures which support rapid intersection of non-convex objects, as well as mesh refinement criteria. The grid generation process is greatly simplified by a preprocessor (described in [Aftosmis, et al.

(98)]) which robustly extracts the wetted surface of a configuration described by surface triangulations of an arbitrary number of individual (possibly intersecting) objects.

Another important aspect of Cartesian mesh methods concerns the numerical issues that arise in computing steady flow. The lack of grid smoothness at the cut-cells where the solid body intersects the grid locally degrades scheme accuracy. However, much work has been done in the last few years developing methods for these types of irregular grids [Coirier and Powell (94)], [Berger and Melton (94)], [Johansen and Colella (99)]. The situation for time dependent flows however is more difficult and less developed. This is mainly due to stability problems encountered by explicit methods on cut-cells whose volumes are potentially orders of magnitude smaller than the regular flow cells.

In both the steady and time dependent cases, mesh refinement is an important component. Section 2.3 examines a generalization of the refinement criteria to include anisotropic subdivision. This approach helps reduce the number of cells produced at each adaptation by a constant factor, and offers the potential of resolving a given geometry with considerably fewer Cartesian cells. Finally, section 4 mentions extensions currently underway to inviscid flows with moving geometry and to the automatic generation of viscous grids with boundary layer zoning.

2 Cartesian Grid Generation Algorithm

The starting point for the grid generator is a closed watertight surface triangulation describing the geometry. In broad outline, our algorithm first intersects the geometry with the mesh, refining the flow field cells where necessary. This process repeats until the maximum specified number of levels has been reached or no more refinement is needed. Only at this point is the detailed geometric information at the cut-cells computed. This is described in section 2.1. As a result of this process, some cut-cells may be split into several separate polyhedra. We describe a robust algorithm for treating such "split-cells" in section 2.2. Finally, section 2.3 presents an extension of the grid generator that allows anisotropic refinement of Cartesian cells. While this approach cannot be applied to boundary-layer zoning, it can substantially reduce the total number of cells required for a given simulation. Since the grid generator views the mesh as a completely unstructured collection of hexahedra, the data structures easily accommodate such anisotropically refined cells.

2.1 Overall Description

Grid generation proceeds in two steps. In the first, a uniform, coarse Cartesian mesh (possibly a single "root" cell) is repeatedly refined using only simple geometric criteria. The goal is to provide a good starting mesh for the flow

solver, and to ensure the geometry is well resolved on the mesh. During this step, the only geometric information needed is whether or not a Cartesian cell is in the flow field, within the solid object, or cut by the geometry. At the end of this step, the flow field cells in the volume mesh have been completely determined, and can be written to disk. This frees active memory so that more elaborate data structures can be retained for determining the precise geometry of the cut-cells without increasing the maximum memory needed during mesh generation. In the second step, the detailed intersection of the geometry and the cut-cells is computed.

In a little more detail, the grid generation starts with an initial coarse mesh of specified dimensions. The cells are classified as flow, cut or solid. Cut-cells can be classified by intersecting them with the surface triangulation. This can be done rapidly using the ADT [Bonet and Peraire (91)] to store the triangles, so that each query takes $O(\log(N_T))$, where N_T is the number of triangles in the surface description. Further speedup comes from using bounding box filters preceding the ADT search. "Flow" and "solid" cells can be classified by ray-casting [O'Rourke (94)] from the test cell to just outside the bounding box of the closed, watertight, geometry. Again the ADT is useful here for searching through surface triangles. However, to minimize this expensive ray-casting procedure, once a cell's status is determined, it shares this status with all its uncut neighbors in a "painting" step. Thus, one ray cast can classify a large number of cells. (These search and traversal algorithms are described in more detail in [Aftosmis, et al. (98)]). Cartesian cells which lie completely internal to the geometry ("solid" cells) are immediately removed from the mesh, and only the flow and cut-cells are retained.

Based on estimates of the surface curvature, cut-cells are marked as needing refinement. A number of neighboring cells are also marked to be refined. These "buffer" layers typically extend three to five cells in all directions. The refinement criteria looks at both the variation of the surface normals within a cell, and the variation of the average normal between cells. This criteria alone would always refine cells that are split by the geometry, even if two distinct surfaces within a cell were planar. To avoid this, the variation of normals for each connected piece of the surface needs to be measured separately. Note that the refinement in the mesh generator is solely based on resolving the geometry. Further solution-adaptive mesh refinement will occur as necessary in the flow solver.

After the cells are marked for division the actual refinement takes place, and new cells and faces are created. To prevent adjacent cell sizes from differing by more than one level, the cell division routines sweep from coarsest to finest. The faces are kept sorted in $x/y/z$ order. To maintain this convention, the face lists are repacked after each complete division pass. Finally, some of the newly refined cells will lie entirely within the geometry. Again, ray-casting determines the cell status of a newly created non-intersected cell. The solid

cells with their associated faces are then deleted from the mesh and the cell and face lists are compacted.

In the second phase of the grid generation, the detailed intersection of the surface triangles and cut cells is computed. The data structures for these cells are necessarily more complex than those of non-intersected Cartesian cells. Each cut-cell maintains a list of its intersected surface triangles. The "flow" faces of the cut cell are determined by clipping the Cartesian faces against the triangles. Each triangle may also be clipped into a convex polygon by the cell that owns it. The true centroids and areas of these polygons, cut faces and cut cells are also computed and retained. Cells and faces in the volume mesh (uncut cells) require 9 four-byte words of storage per cell. Since each cut-cell is linked to its triangles, the storage for cut cells is problem dependent. However, in the example of figure 5 the cut cells generated using 9 levels of mesh refinement required an average of 84 words per cell. This includes full information, such as the double precision centroids of each triangle's intersection with each cut cell. (On average each cut cell was linked to 4.19 triangles.) Since the volume mesh is written to disk and storage is released before the cut cell processing, the maximum memory used to generate the mesh was 53 Mb.

2.2 Split-Cells

As with any method, certain annoying details must be taken care of. For Cartesian grids with embedded geometry, this laborious chore comes from cut Cartesian cells which are "split" into multiple distinct "flow" regions by the geometry. For example, a thin wing may bisect the cells for the trailing 15% of the chord, even though the Cartesian cell size would be sufficient to resolve either the top or bottom of the thin surface alone. It can be quite inefficient to use mesh refinement alone to resolve the entire thin piece of geometry. Even geometry that does not appear thin may produce a small number of split-cells. Figure 1 shows examples of split-cells stemming from two common geometric situations.

The main difficulty in treating this problem is recognizing that a cell has been split, and connecting the various faces with their corresponding (possibly split) neighbor cells. This is further complicated by the possibility of refinement boundaries occurring at such faces. A coarse cell face may be split, whereas the corresponding face on the refined neighbor may not be. The example in figure 2 shows a large cell a linked to four smaller cells. The high x face of cell a is split, but it links to the low x face of cell b which is not. In the case presented by the figure, five faces would be stored, one for ab, two for ac (cell c is split), one for ae and one for ad. To include all diabolical cases, the algorithm also permits cells to be split into more than two flow cells.

Our treatment of split-cells first groups the connected surfaces within a cell into distinct polyhedra. Since phase two of the mesh generator works cell-

by-cell, each cell must save enough information to correctly match with its neighboring possibly split-cells as a post-processing step. We match fragments of the Cartesian edges themselves. Each split-cell or face must have an edge fragment from the Cartesian mesh with an original Cartesian edge. Since Cartesian edges are known by their *integer* coordinates, they can be used to unambiguously match the fragments on either side. Relying on constructed geometry, (for example resulting from the polygon clipping) may introduce floating-point errors which could lead to non-robustness. Centroid matching algorithms can be fooled. Using an integer-based approach eliminates any dependence on floating point comparisons.

2.3 Anisotropic Refinement

One of the major drawbacks of Cartesian methods is its lack of flexibility, particularly in the ability to do directional refinement. A solution to this problem would be a breakthrough for extending Cartesian mesh methods to the high Reynolds number viscous case. An interesting first stab at this problem is in [Coirier (94)], where cells in a two dimensional quadtree mesh are refined with only one planar cut rather than the usual two, and the direction is arbitrary. Unfortunately, this produced meshes that were too irregular for the flow solver to compute an accurate solution. Here we explore the usefulness of a limited anisotropic refinement capability along Cartesian coordinate directions. The asymptotic complexity of this approach is not sufficient to produce "boundary layer" zoning for viscous flows; our aim is to reduce the total number of cells in an inviscid mesh. At this point we are only considering the use of anisotropic refinement to resolve static geometry. We follow the same procedure as in [Aftosmis (94)], although that paper applied it to a three dimensional flow field without geometry.

The main difficulty lies in establishing useful criteria for directional refinement, and trying to insure that there are smooth transitions between anisotropic regions in different directions. Our strategy is two-fold: if the variation of surface normals within a cell is large enough to warrant refinement, and is almost completely in one coordinate direction, we refine only in that direction. Our definition of "almost completely" is illustrated in figure 3, where if the variation in the surface normals is outside an angle δ of the coordinate direction, refinement in that direction occurs. Secondly, we use the cautious strategy of also refining in the direction normal to the surface normal, or most parallel to the geometry. The surface normals do not give any information about this direction. The expectation is that the flow field associated with surface curvature will warrant the additional refinement. Note that with this strategy, no cell will be refined in only one direction.

With anisotropic refinement, it is more difficult to ensure a smooth variation in the mesh. In addition, one has to be careful that at least one of the cells adjacent to the face has the correct picture of it. For example, consider an x-face. If the cell on the left is refined in y and the cell on the right is

refined in z, then neither cell has a correct picture of the face. To prevent such anomalies, each cell "shares" its refinement tag with its neighbor. Since this could propagate, invalidating previously acceptable faces, several sweeps over the face list may be necessary. Although a diabolical case may need a large number of sweeps, in practice one or two passes suffices.

The refinement strategy described above for uniform isotropic refinement is actually implemented by sweeping over the 3 coordinate directions individually. Each refinement bisects the cell in one direction only. Each new face that bisects a cell creates one new cell. Both the new and old cells adjust their locations (encoded into unique cell "names" [Aftosmis, et al. (98)]). The faces that point to these cells must also be adjusted. Since we do not keep cell-to-face pointers, this is accomplished by keeping a single pointer from old cells to their new sibling. A sweep over faces easily adjusts the affected cells. Directional refinement is easily implemented within this framework.

As a model geometry for anisotropic refinement consider a cylinder with its axis aligned with the z-axis. Intuitively, one expects refinement in the x and y directions, since the cylinder has a circular crosssection in the x-y plane. However, there is no curvature in the z direction. Table 1 compares the number of cells in the isotropically refined mesh with the corresponding anisotropic mesh with the same number of levels as a function of the maximum allowed aspect ratio. The initial coarse mesh is $24 \times 24 \times 24$ cells, the refinement criteria uses a variation in the surface normals of 5 degrees, and the buffer zones are three cells wide. Refinement occurs along the length of the cylinder anisotropically, but the criteria triggers isotropic refinement at the corners. Figure 4 shows an aspect ratio 8 mesh around the cylinder. The growth rate in the number of cells is substantially reduced in the higher aspect ratio case, with the total number of cells a factor of 2.2 less with aspect ratio 4, and 2.7 with aspect ratio 8.

Ref. Level	# Cells (AR 1)	# Cells (AR 2)	# Cells (AR 4)	# Cells (AR 8)
1	32060	25552		
2	86640	59822	49046	
3	260460	162226	116428	97818

Table 1. Comparison of number of cells using isotropic refinement versus anisotropic refinement, as a function of the maximum allowable aspect ratios.

For a more realistic test case, we compare results for an ONERA M6 wing with anisotropic refinement. The table below shows the total number of cells in the isotropic mesh, and aspect ratio 2, 4 and 8 meshes. Five buffer cells were used. A cell was tagged for refinement if the variation in the components of the normals exceeded 10 degrees, and the directional angle refinement criteria

was 30 degrees. For the results in the table below, the geometry was tagged for isotropic refinement for 6 levels. After that level of resolution, the directional refinement could begin. Figure 5 shows the Cartesian cut cells that intersect the geometry, color coded by aspect ratio. The leading and trailing edges are both refined to the maximum aspect ratio allowed. Even for this case, the savings in total number of cells between the isotropic mesh and the aspect ratio 8 mesh is over a factor of 2.

Ref. Level	# Cells (AR 1)	# Cells (AR 2)	# Cells (AR 4)	# Cells (AR 8)
7	163618	116835		
8	328963	217638	167206	
9	703163	447363	320843	246220
10	1469067	929389	649684	497540

Table 2. Comparison of number of cells using anisotropic refinement for the OneraM6 wing.

3 Computational Results

We use an example of a mesh around a business jet to illustrate the run-time performance of the grid generator. There are 92850 triangles in the initial surface description. The final mesh has 9 levels of refinement, with 2.29 million hex cells and 7.3M faces. Only 310K of the cells (13% of the mesh) intersected the geometry. There were 17827 split polyhedral cells in the mesh, despite the number of levels of refinement.

The total number of cells in the mesh after each level of refinement is shown in Table 3. As the mesh refines, the algorithm monitors the trend of the cell growth factor. A curve fit is used to predict the number of cells in the final mesh. This information makes it possible to minimize the number of memory re-allocations and copies required during the mesh generation process. For example, this growth factor for the last 3 refinements, as shown in Table 1, is 2.20, 2.18, and 2.14 A buffer zone of 3 cells was used in this mesh. The refinement criteria was 10 degrees. Six levels of mesh were refined around the body before the algorithm started estimating where to refine.

The complete grid generation procedure, including the calculation of the face and cell centroids, took 3 minutes 51 seconds on an SGI R10000 workstation. A maximum of 308 Mb was used to generate the 9 level mesh. Figure 6 shows 10 cutting planes through the mesh, which provide a good indication of the refinement criteria's behavior. For comparison, the anisotropic 9 level mesh, with a maximum aspect ratio of 8, had 1225022 cells.

Ref. Level	Total # Cells	# Cut Cells	Total # Faces
5	81429	4603	256943
6	222714	16924	700969
7	490237	52139	1559113
8	1069985	144282	3416549
9	2291286	309981	7289082

Table 3. Growth of number of cells with refinement level for business jet example.

4 Future Research

Two projects currently underway would greatly extend the capabilities of Cartesian mesh methods. The first computes inviscid flow with moving geometry, including the case with geometry components in relative motion. The difficulty here is in developing numerical methods to handle the case where Cartesian cells are newly uncovered during a time step, or become completely covered by the geometry. Some two dimensional work in this direction is in [Bayyuk, et al. (96)]; a new approach is being developed by [Forrer and Berger (98)].

The second extension is to try to use Cartesian meshes to help generate viscous meshes with boundary layer zoning in an automatic way. As contrasted with previous efforts by [Coirier (94)], the new approach [Delanaye et al. (99)] uses a prismatic body-fitted grid, with a background Cartesian grid. However, to reduce the dependence of the body-fitted grid on the quality of the surface triangulation, a new triangulation based on the intersection of the original surface with the Cartesian grid is used to spawn the new grid. Since this new approach automatically generates the surface discretization used by the flow solver, it maintains the *decoupling* between the geometry description and surface mesh seen by the flow solver. This decoupling is one of the biggest advantages of Cartesian meshes for inviscid flows, and retaining it in the viscous case is an important first step.

Acknowledgements

The authors thank Eric Charlton for the use of his business jet surface triangulation. Marsha Berger was supported in part by DOE Grant DEFG02-92ER25139 and by AFOSR Grant F49620-97-1-0322. Some of this work was performed while at RIACS, whose support is gratefully acknowledged.

References

Aftosmis, M., Upwind Method for Simulation of Viscous flow on Adaptively Refined Meshes, AIAA J. 32:2, Feb., 1994.

Aftosmis, M., Berger, M. and Melton, J., Robust and Efficient Cartesian Mesh Generation for Component-Based Geometry, AIAA J. 36:6, June, 1998.

Aftosmis, M., Melton, J. and Berger, M., Adaptation and Surface Modeling for Cartesian Mesh Methods, AIAA paper 95-1725, June, 1995.

Bayyuk, S., Powell K. and van Leer, B., An Algorithm for Simulation of Flows with Moving Boundaries and Fluid-Structure Interactions, Proc. 1st AFOSR Conf. on Dynamic Motion CFD, June, 1996.

Berger, M. and Melton, J., An Accuracy Test of a Cartesian Grid Method for Steady Flow in Complex Geometries, Proc. 5th Intl. Conf. Hyp. Problems., Stonybrook, NY, June, 1994.

Bonet, J. and Peraire, J., An Alternating Digital Tree (ADT) Algorithm for Geometric Searching and Intersection Problems, Intl. J. Num. Meth. Engg. 31:1-17, 1991.

Charlton, E. and Powell, K. , An Octree Solution to Conservation Laws Over Arbitrary Regions (OSCAR), AIAA paper 97-0198, Jan., 1997.

Coirier, W., An Adaptively Refined, Cartesian, Cell-Based Scheme for the Euler and Navier-Stokes Equations. PhD Thesis, University of Michigan, 1994.

Coirier, W. and Powell, K., An Accuracy Assessment of Cartesian-Mesh Approaches for the Euler Equations, AIAA paper 93-3335, July, 1993.

Delanaye, M., Aftosmis, M., Berger, M., Liu, Y. and Pulliam, T., Automatic Hybrid-Cartesian Grid Generation for High-Reynolds Number Flows Around Complex Geometries. Submitted to AIAA, Reno, Jan, 1999.

Forrer, H. and Berger, M., Flow Simulations on Cartesian Grids involving Complex Moving Geometries, Proc. 7th Intl. Conf. Hyp. Problems, Zurich, Switz., Feb. 1998.

Johansen, H. and Colella, P., A Cartesian Grid Embedded Boundary Method for Poisson's Equation on Irregular Domains. To appear, J. Comp. Phys., 1999.

Karman Jr., S., SPLITFLOW: A 3D Unstructured Cartesian/Prismatic Grid CFD Code for Complex Geometries, AIAA paper 95-0343, Jan., 1995.

Landsberg A. and Boris, J., An Efficient Method for Solving Flows around Complex Bodies, NRL Review, 1993.

Melton, J., Automated Three-Dimensional Cartesian Grid Generation and Euler Flow Solutions for Arbitrary Geometries. PhD Thesis, U.C. Davis, June, 1996.

O'Rourke, J., Computational Geometry in C. Cambridge Univ. Press, 1994.

Pember, R.B., Bell, J.B., Colella, P., Crutchfield, W.Y. and Welcome, M.W., Adaptive Cartesian Grid Methods for Representing Geometry in Inviscid Compressible Flow. AIAA paper 91-1542, June, 1991.

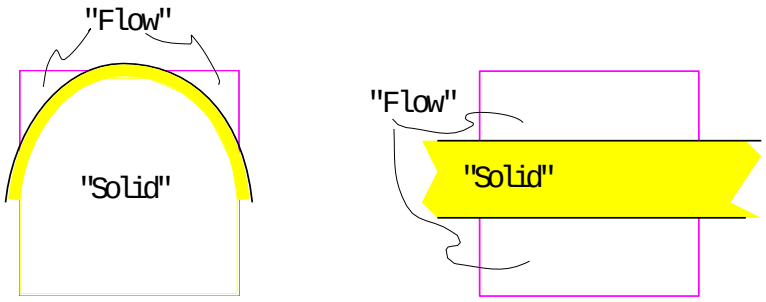

Fig. 1. Two common examples of "split-cells" where Cartesian hexahedra are cut into multiple distinct flow-through regions.

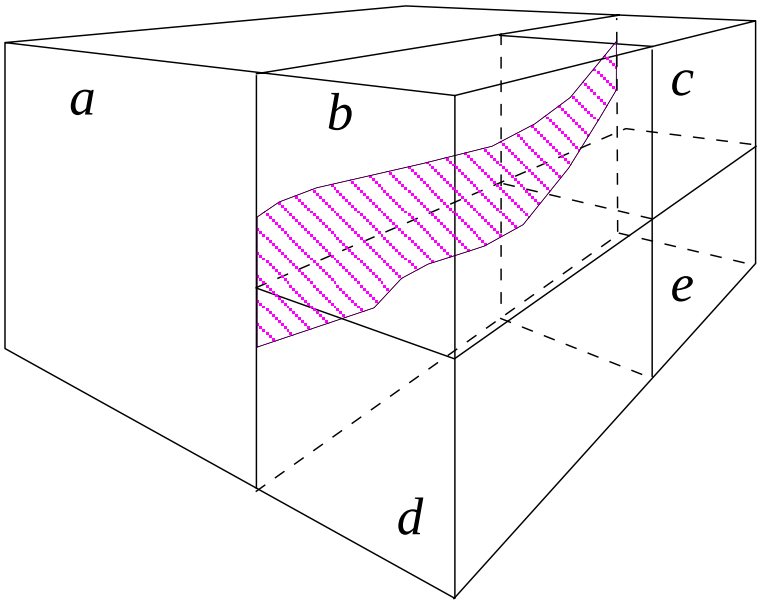

Fig. 2. A general split face matching algorithm has to treat faces where the cell on one side only is refined. The high *x face of cell* a is split into two face polygons which must be matched with the 5 polygons on the low *x face of the small cells. The algorithm stores five faces for this case: one for* ab, two for *ac (cell* c is split), one for *ae and one for* adAll 4 faces on the refined side are split into 8 regions; the coarse side is split into two pieces. The algorithm matches fragments of the face polygons which lie on Cartesian edges.

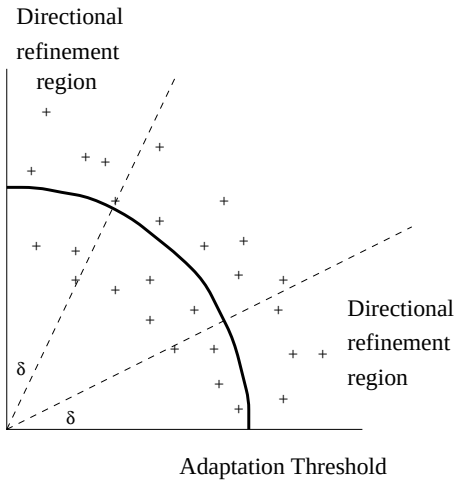

Fig. 3. The "+" signs illustrate the maximum variation in surface normals for a two dimensional example. If the magnitude of the variation exceeds the adaptation threshold, the cell needs to be refined. If the variation is all in one direction (within the slivers marked by the dotted lines), the cell can be refined anisotropically.

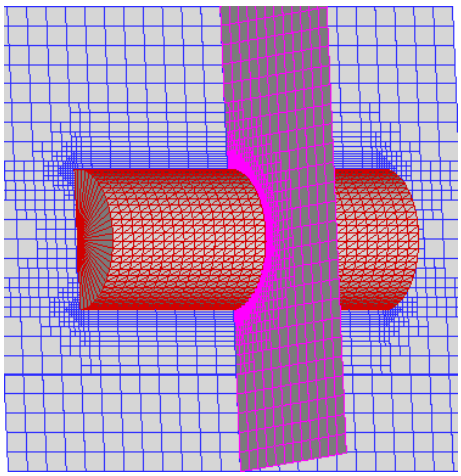

Fig. 4. Mesh around a cylinder with anisotropic refinement, aspect ratio 8.

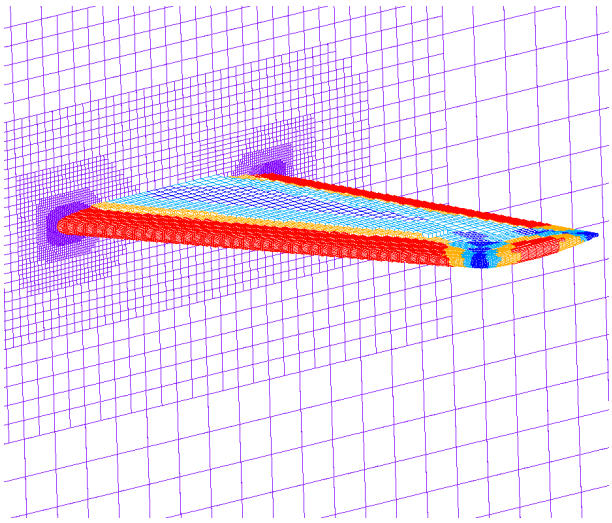

Fig. 5. Aspect ratio 8 mesh for the ONERA M6 wing.

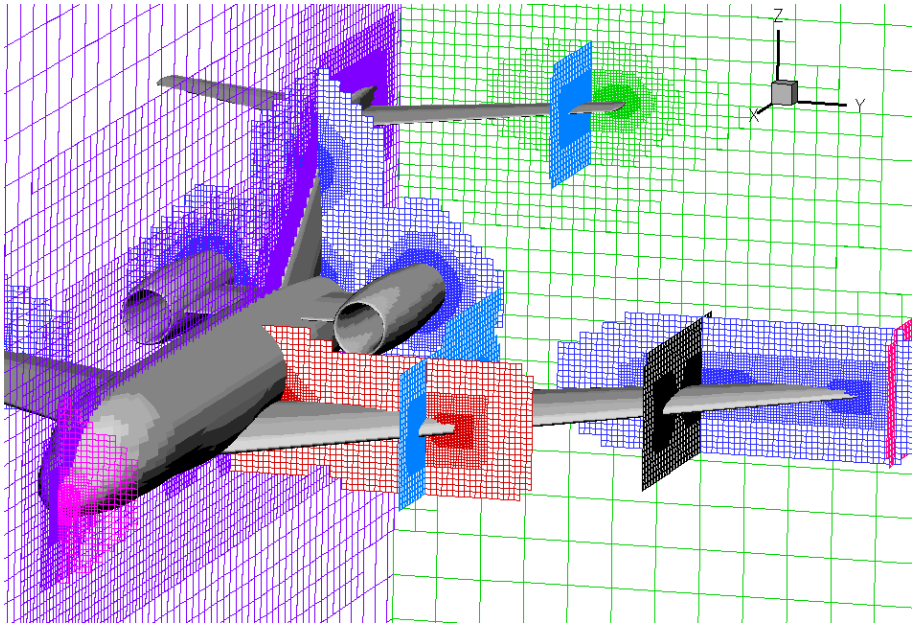

Fig. 6. Cartesian grid for business jet.